Bigtable: A Distributed Storage System for Structured Data

By Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

Presented at 7th USENIX Symposium on Operating Systems Design and Implementation on November 7, 2006

Mason Hudson, David Heineman, Ishaan Immatty, Jacob King, Vince Ly

Introduction

Problems Solved

- Management of structured data at a very large size
  - Designed to scale
  - Petabytes of data across thousands of commodity servers
- Wide applicability, scalability, high performance, and high availability
- Providing clients with a straightforward data model that enables dynamic control over data layout and format, while allowing clients to reason about the locality properties of the data represented in the underlying system

Why It Matters

- Over sixty Google products/projects have these demands
- These applications have diverse demands for Bigtable, both in terms of data size and latency requirements

https://play.google.com/store/apps/details?id=com.google.android.apps.giant&hl=en

https://w7.pngwing.com/pngs/611/112/png-transparent-google-earth-google-maps-google-thumbnail.png

Data Model

Overview

- A Bigtable is a sparse, distributed, persistent multidimensional sorted map

- The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes

(row:string, column:string, time:int64) → string

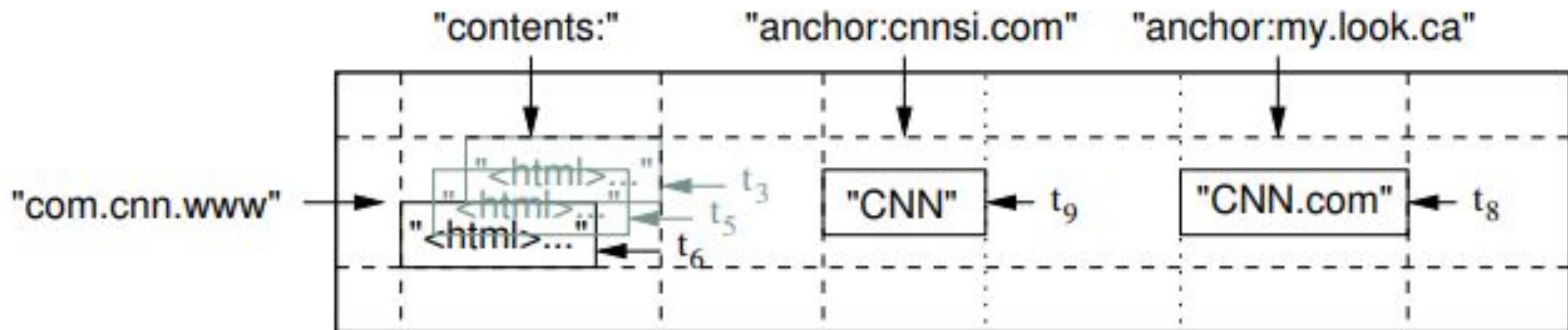


Figure 1

Rows

- Row keys in a table are arbitrary strings
  - Up to 64KB in size, but 10–100 bytes is a typical for most users
- Each read/write of data under a single row key is atomic
- Ordered lexicographically
- Each row range is called a *tablet*
  - The unit of distribution and load balancing

"contents:" "anchor:cnnsi.com" "anchor:my.look.ca"

"com.cnn.www" "<html>..." t_3 t_5 t_6 "CNN" t_9 "CNN.com" t_8

Figure 1

Column Families

- Column keys grouped into sets—form the basic unit of access control
- Must be created before data can be stored under any column key in that family
- Named using the following syntax: *family:qualifier*
- Access control and both disk and memory accounting are performed at the column-family level

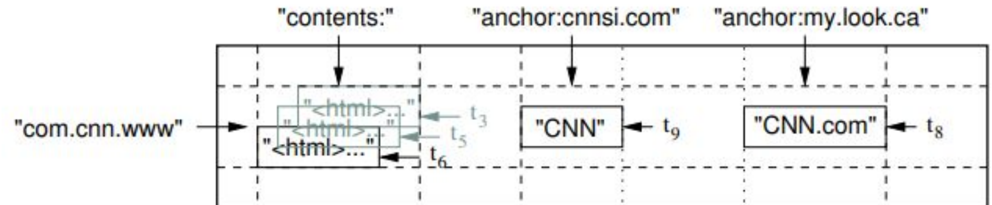


Figure 1

Timestamps

- Each cell in a Bigtable can contain multiple versions of the same data
    - Versions indexed by timestamp (64-bit integers)
- Assigned by Bigtable or assigned by client applications
- The client has the option to retain either only the latest n versions of a cell or all versions until they are older than n seconds

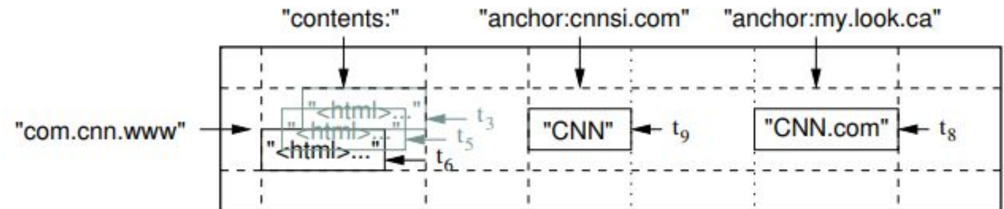


Figure 1

BigTable API

Designing the API

- Built to satisfy *most* DB use cases across Google to be easily customizable
- The API provides:
  - Creating, deleting tables and column families
    `DeleteCells(), DeleteRow()`
  - Functions for changing cluster, table and column family data
    `Set()`
  - Reads!
    `Scanner`
  - Access control management

Client API: Modifying

- BigTable offers a convenient API which allows setting or deleting values by defining, then running operations
  - Write/delete values
  - Iterate through subset of table rows

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");
// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

Client API: Reading

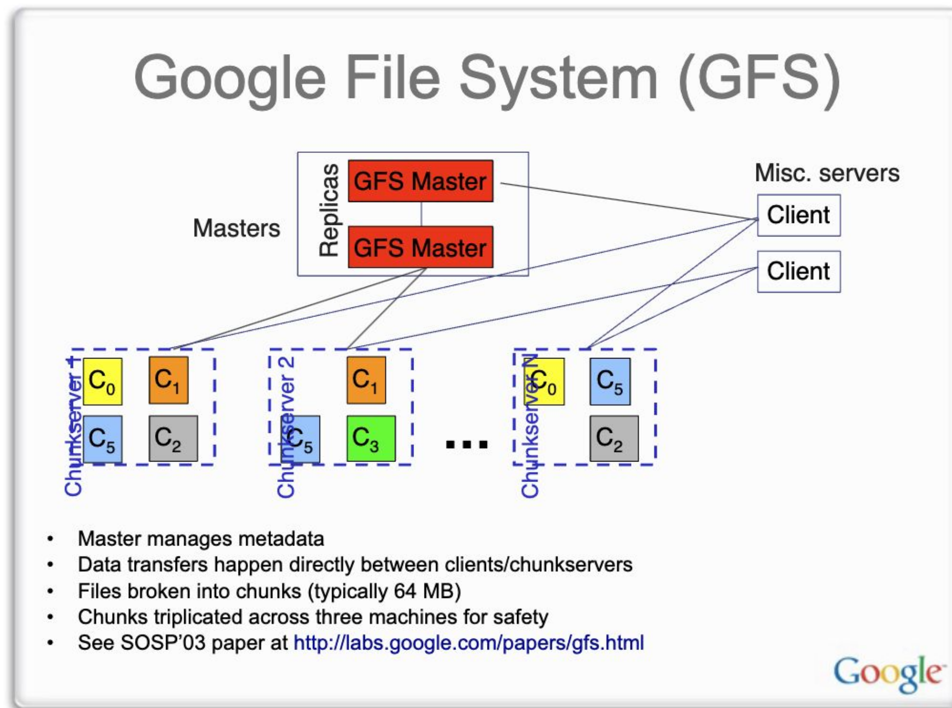
- BigTable supports **single-row transactions** which perform atomic r/w operations on sequences.
- **Sawzall** allows client-side scripts to modify data as part of a Map-Reduce job. Sawzall has since been discontinued for different features.

```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
          scanner.RowName(),
          stream->ColumnName(),
          stream->MicroTimestamp(),
          stream->Value());
}
```

Building Blocks

Data Storage (GFS)

- GFS contains arbitrary files, with a master containing names and metadata
  - Stored in `SSTable` format (64 KB each)
  - On retrieval, binary search over the master is used to find the relevant `SSTable`
- BigTable uses GFS to run queries in a MapReduce-like setup
  - Allows for a fast, persistent datastore



Locking (Chubby)

- BigTable relies on the **Chubby** lock service, which assumes a high availability and persistent distributed lock system
  - Chubby has **five active state replicas** and communicate with each other using the **Paxos algorithm** to ensure consistency
  - If the client fails to continuously renew its session with Chubby, any locks the client has expires and are dropped.
- If Chubby is unavailable, access to the BigTable becomes unavailable.
  - In their experiments, Chubby downtime causing BigTable downtime accounted for **0.0047% of total availability**.

Implementation

Tablets and Tablet Location

- Tablets are the unit of distribution and load balancing, comprised of ranges of contiguous rows, each around 100-200 MB
- A Bigtable implementation has three components:
    - A library linked to every client
    - A master server responsible for load balancing and maintaining tablet servers
    - Many tablet servers that manage a set of tablets (10 -1000 typically) and handle read and write requests
- Tablet location information is stored as shown in Figure 4
    - METADATA tables cannot split and have the UserTable's table identifier and end row as well as secondary information such as access logs
    - Each METADATA row is about 1KB, with a limit of 128MB tablets, this scheme can address 2^34 tablets or 2^61 bytes (~2000 PB)
    - Tablet locations are cached on the client library

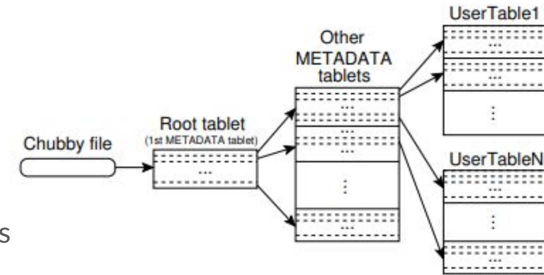


Figure 4: Tablet location hierarchy.

Tablet Assignment and Server Details

- When a tablet is unassigned, the master server assigns the tablet to a server with sufficient room
- The master server can instantiate tablet creation, deletion, and merges
- Tablet servers can split tablets, and will update the METADATA table and the master server
- When a new tablet server starts:
  - A tablet server acquires a lock on a uniquely named file in a Chubby servers directory
  - The master server monitors this directory to find tablet servers and periodically checks the locks on the files
  - If the tablet server loses its lock or is unreachable, the master removes the server file and reassigns its tablets
- When a new master server starts:
  - The master server grabs a master lock in Chubby and scans the server directory to find live servers
  - The server communicates with every tablet server to see what tablets are assigned where
  - The server scans the METADATA table to find any unassigned tablets.

Tablet Serving

- The persistent state of a tablet is stored in Google File System (GFS)
- Recently committed updates are stored in memory in a sorted buffer called a memtable
- Older updates are stored in SSTables, which are kept track of by the METADATA table
- Write operations are validated, authorized using a list of authorized writers from a Chubby file, and added to the tablet. Commits are grouped to improve throughput
- Read operations are validated, authorized, and executed using a merged view of the memtable and SSTables

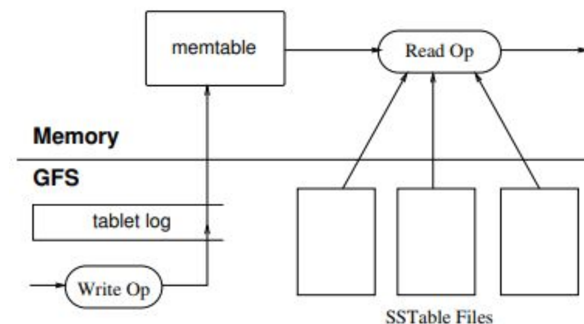


Figure 5: Tablet Representation

Compactions

- Bigtable uses three types of compactions:
  - Minor: When the memtable reaches a certain size, a new memtable is created and the old table becomes an SSTable and added to GFS
  - Merging: If there are too many SSTables, a few SSTables and the memtable can be merged into one SSTable
  - Major: A merging compaction that merges rewrites all SSTables into one
- Major compactions remove deleted data, since the other compactions can still have deletion entries for data that still exists in older SSTables

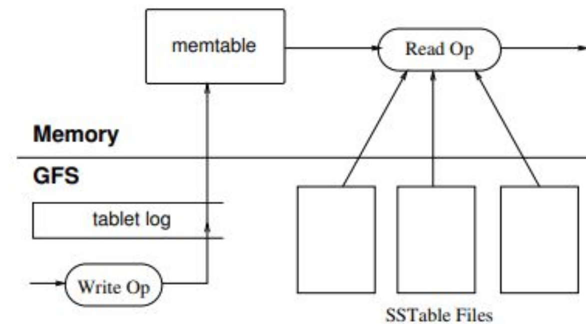


Figure 5: Tablet Representation

Refinements

Refinements

- Locality groups
  - Column families can be grouped into a locality group, allowing more efficient data reads
  - An SSTable is generated for each locality group in a tablet
  - Also allows for some columns to be loaded in memory at all times
- Compression
  - Users can specify a compression scheme for SSTables for locality groups
  - These algorithms are emphasized for speed rather than space compression
- Caching for read performance
  - Tablet servers use two levels of caching:
    - Scan cache has key-value pairs returned by the SSTables
    - Block cache caches SSTable blocks that were read from GFS, helping save reads on contiguous data

Refinements

- Bloom filters
  - A bloom filter for SSTables allow the program to more efficiently find a certain piece of data
  - Read operations require reading from all SSTables, so bloom filters reduce disk accesses
- Commit-log implementation
  - Each tablet server has a single commit log, rather than a single commit log for each tablet, reducing the total number of writes to GFS
  - If a server fails, log entries are sorted in order of the keys <table, row, log sequence number>, meaning all logs for tablets are now contiguous and can be sent to the server containing each redistributed tablet
- Speeding up tablet recovery
  - When a tablet is moved across servers, it undergoes a minor compaction first so log entries do not need to be transferred as well
- Exploiting Immutability
  - SSTables are immutable, meaning that concurrency control over rows can be implemented efficiently and splitting tablets can be done by letting child tablets use the parent SSTables

Performance Evaluation

Key Observations:

- Scalability
  - More work can be done as more servers are added
- Throughput Variability
  - Servers individually may do less
    - Due to servers using the same resources
- Efficiency in Reads and Writes
  - Efficiency is dependent on the task
  - Reading data tends to be faster compared to random reads

Main Features

- Memtables and SSTables
  - Special memory areas (memtables) are used to gather data before saving it to disk (SSTables)
  - They are used to write data quickly and organize the disk data efficiently
- Caching Mechanisms
  - Caches are used to remember data that is read frequently
  - Reduces the need to go back to the disk and speeding up reading data
- Data Compression
  - Squeezes the size of the data in SSTables
  - Bigtable needs less space and can read and write data quickly
  - Space and speed efficient
- Load Balancing
  - Bigtable spreads out the work evenly across its servers
  - Overloads are avoided
- Bloom Filters
  - Checks if certain data is not on the disk
  - Makes reading data faster by avoiding unnecessary checks

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

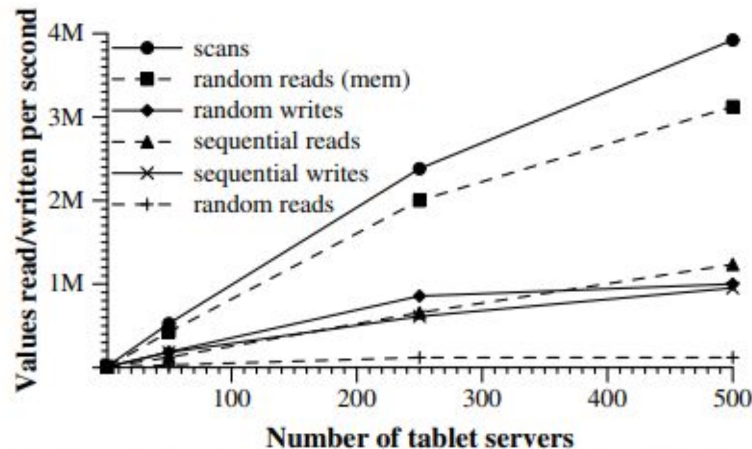


Figure 6: Number of 1000-byte values read/written per second. The table shows the rate per tablet server; the graph shows the aggregate rate.

- Random Reads: As random reads per server decreases, number of servers increase (1212 on single server -> 241 on 500 servers)
    - Random Reads (mem): more of a gradual decrease in reads per servers
- Random Writes: As random writes per server decreases, number of servers increase (8850 on single server -> 2000 on 500 servers)
- Sequential Reads/Writes: Operations are less severe compared to random as servers increase (read: 4425 -> 2469, write: 8547 ->1905)
- Scans: Number of per-server performance decreases as number of servers increase (15385 on single server -> 7843 on 500 servers)

Real Applications

Google Analytics

- Provides info about:
  - Website traffic
  - Visitor statistics
  - Site tracking
- Bigtable stores and process this data
  - Primary Tables: session data & summary table
- Ideal for managing big datasets

Google Earth

- Provides satellite imagery of the world
- Uses Bigtable:
  - For preprocessing imagery data
    - Stores raw imagery in a table
  - Serving imagery to clients
    - Indexes to serve satellite imagery by using Bigtable’s scalability

Personalized Search

- Tailors search results based on search history
- Bigtable
    - Stores user queries & clicks
    - Users are assigned to a row through user ID
- Handles data analysis and personalized algorithms

Lessons

Issues Found & Lessons Learned

- Large distributed systems are vulnerable to many types of failures, not just the standard network partitions and fail-stop failures found in many distributed protocols
    - Memory and network corruption, large clock skew, hung machines, extended and asymmetric network partitions, bugs in other systems, overflow of GFS quotas, and hardware maintenance
    - Addressed by changing various protocols & removing assumptions made by one part of the system about another part
- Delay adding new features until it is clear how the new features will be used
- Importance of proper system-level monitoring (monitoring Bigtable itself as well as the client processes)
    - Makes it easier to find issues and resolve them
- Value of simple designs
    - Imperative for code maintenance and debugging

Related Work

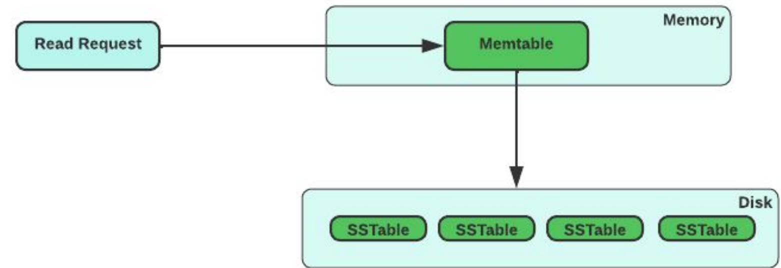
Important Works

- Maccormick, J., Murphy, N., Najork, M., Thekkath, C. A., And Zhou, L. Boxwood: Abstractions as the foundation for storage infrastructure. In Proc. of the 6th OSDI (Dec. 2004), pp. 105–120.
  - Boxwood provides infrastructure for building higher-level services such as file systems or databases, while Bigtable directly supports client applications that wish to store data
- ORACLE.COM. www.oracle.com/technology/products/-database/clustering/index.html. Product page.
  - Oracle's Real Application Cluster database uses shared disks to store data (Bigtable uses GFS) and a distributed lock manager (Bigtable uses Chubby)
- Baru, C. K., Fecteau, G., Goyal, A., Hsiao, H., Jhingran, A., Padmanabhan, S., Copeland, G. P., And Wilson, W. G. DB2 parallel edition. IBM Systems Journal 34, 2 (1995), 292–322.
  - IBM's DB2 Parallel Edition is based on a shared-nothing architecture similar to Bigtable

https://www.oracle.com/technetwork/database/options/clustering/rac-twp-overview-5303704.pdf
https://www.ibm.com/products/instana/supported-technologies/db2-monitoring

Important Works

- Greer, R. Daytona and the fourth-generation language Cymbal. In Proc. of SIGMOD (1999), pp. 525–526
  - AT&T's Daytona does vertical and horizontal data partitioning into flat files and achieves good data compression ratios
- O'neil, P., Cheng, E., Gawlick, D., And O'neil, E. The log-structured merge-tree (LSM-tree). Acta Inf. 33, 4 (1996), 351–385.
  - Bigtable uses memtables and SSTables to store updates to tablets like how a Log-Structured Merge Tree stores updates to index data
- Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'neil, E., O'neil, P., Rasin, A., Tran, N., And Zdonik, S. C-Store: A column oriented DBMS. In Proc. of VLDB (Aug. 2005), pp. 553– 564.
  - Both systems use a SN architecture and have two different data structures, one for recent writes, and one for storing long-lived data, with a mechanism for moving data from one form to the other but the systems differ significantly in their API



VERTICA
by opentext™

https://www.geeksforgeeks.org/introduction-to-log-structured-merge-lsm-tree/
https://en.wikipedia.org/wiki/Vertica#/media/File:Vertica-OT.svg

Conclusions

Takeaways

- As of August 2006, more than sixty projects are using Bigtable
- Users like the performance, high availability, and easy scalability provided by simply adding more machines to the system
- New users are sometimes uncertain of how to best use Bigtable
  - Especially if accustomed to using relational databases that support general-purpose transactions
  - Successful use by many Google products demonstrates that design works well in practice
- Additional features being implemented like mentioned before
- Started providing Bigtable as a service to product groups so individual groups do not need to maintain their own clusters
  - As service clusters scale, resource-sharing issues will need to be dealt with
- Many advantages to building Google specific storage solution
  - Flexibility and ability to solve issues as they arise

Bigtable Today

- Available as a public subscription service and supports many of Google's own core services, including Google Search, Google Maps, Google Drive, Google Analytics and YouTube
- Currently manages over 10 exabytes of data
- Well-suited to batch MapReduce operations, machine learning applications, and stream processing and analytics
- Led to creation of systems like Apache HBase database, Cassandra, and Hypertable

https://www.techtarget.com/searchdatamanagement/definition/Google-BigTable
https://dbdb.io/db/cloud-bigtable/revisions/4

Study Questions

1. What are the two sources of committed update records when data is read from a tablet? Why are they separated and what are the benefits of this?
2. Explain the rationale behind Bigtable's data model, particularly focusing on the use of row keys, column families, and timestamps. How does this data model support efficient range queries and time-based data retrieval and what are some drawbacks of this model?